Recursion Theorem; Other Models of Computation

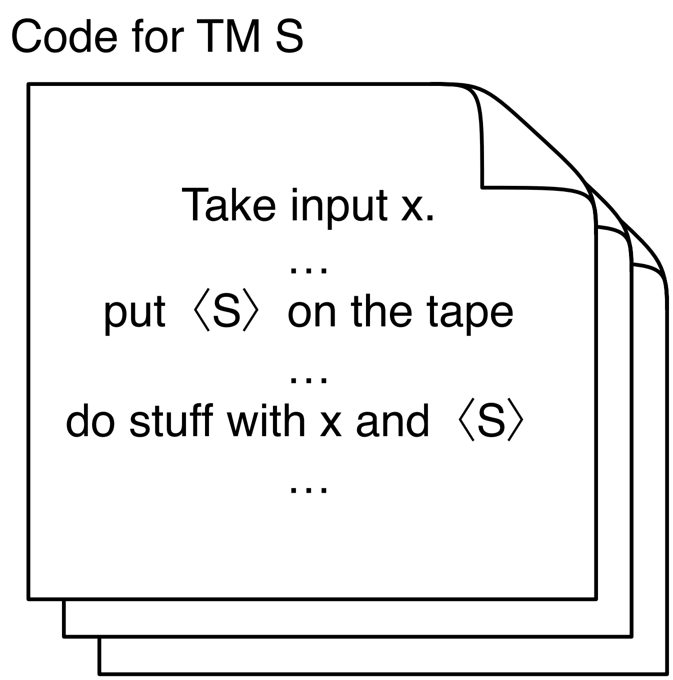
CS 81: Computability and Logic

November 30, 2010

The Recursion Theorem

Recursion Theorem (Kleene 1938)

- **Informally**: A program can have access to its own description (code).



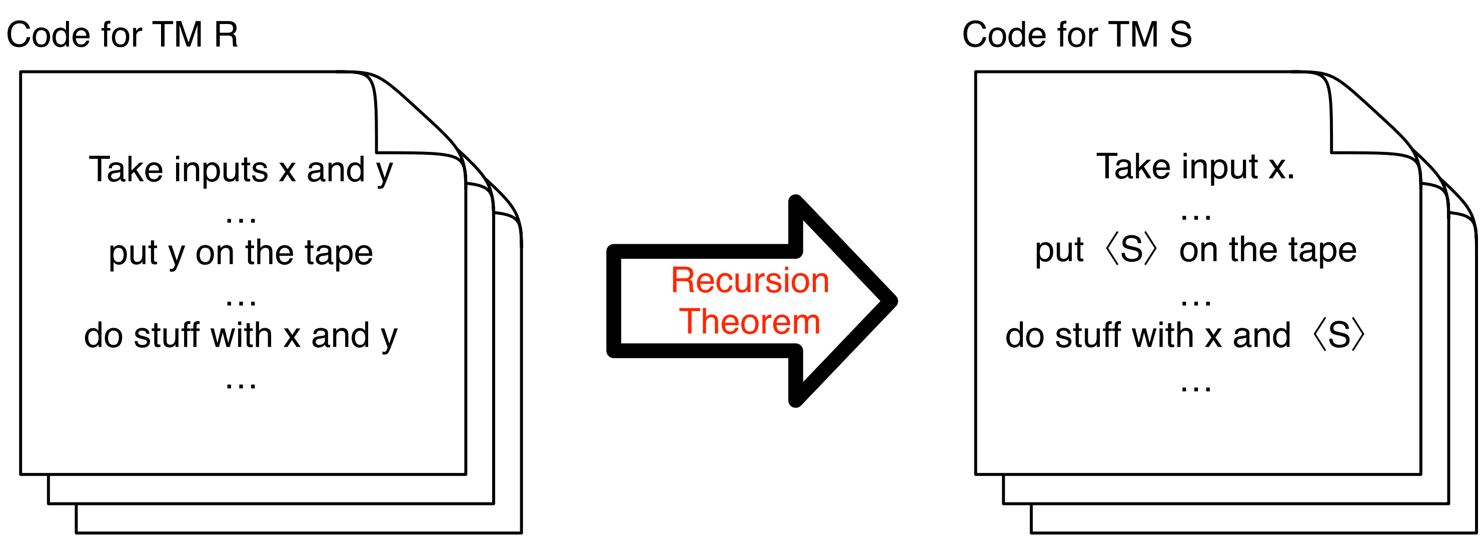
Recursion Theorem (Kleene 1938)

Formally:

If R is a Turing machine computing a binary function R(x, y), then there is a Turing machine S computing a unary function such that:

$$S(x) = R(x, \langle S \rangle)$$

where $\langle S \rangle$ is the description of S itself.



Application: Undecidability of A_{TM} (again)

- Suppose A_{TM} were decidable using TM A.

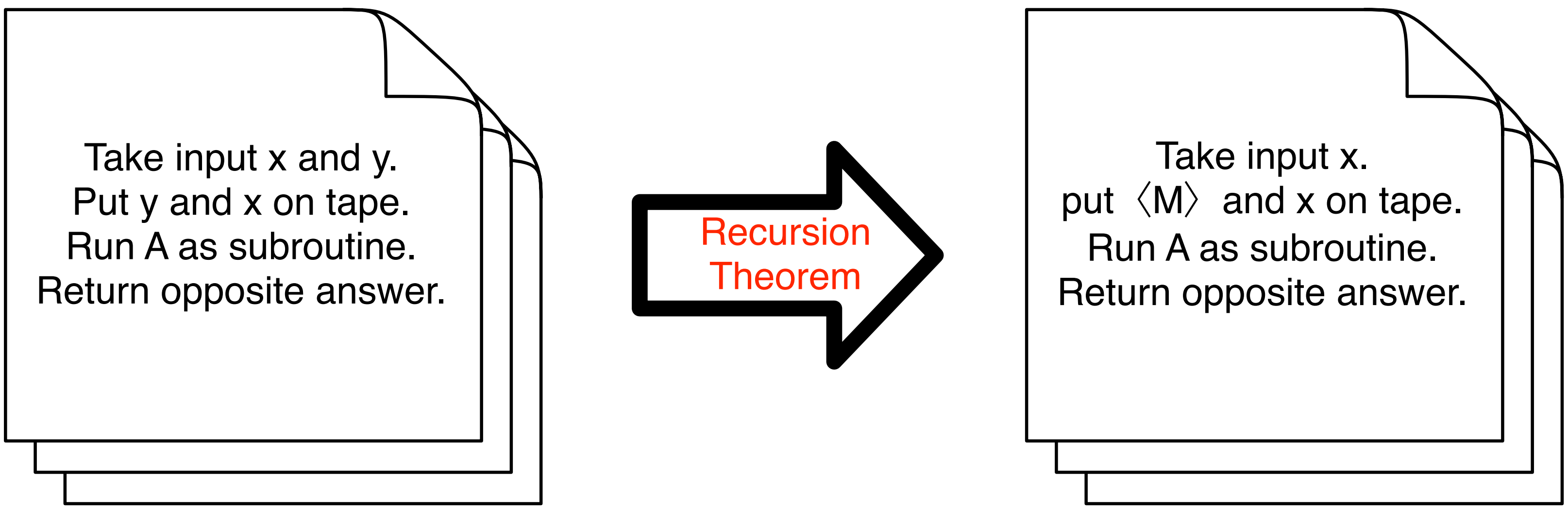
- Define M(x) as follows:
  1. Take input x;
  2. Use A to decide whether M accepts x, i.e., whether
     $\langle M,x \rangle \in A_{TM}$
  3. Return the opposite answer.

- M can't exist, so A must not exist. QED

Recursion Theorem to the rescue

Code for TM R

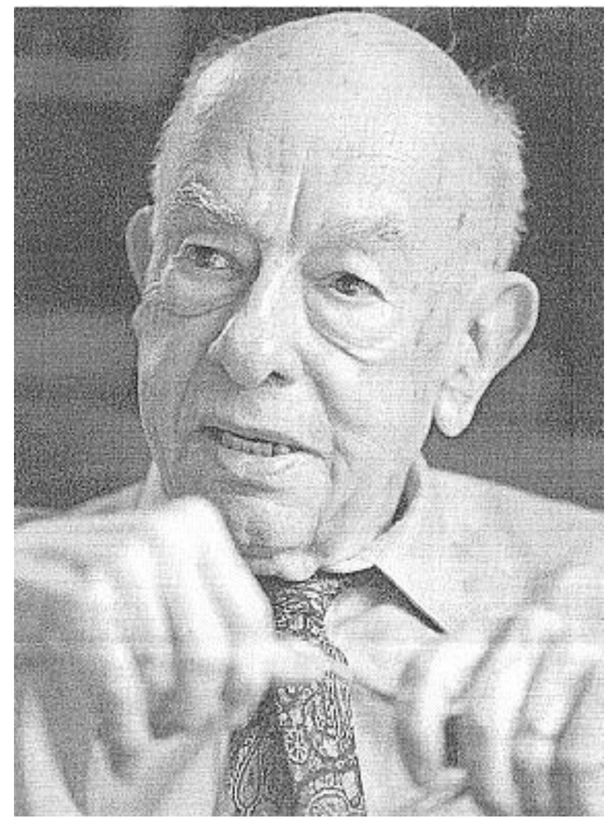
Code for TM M



Self-Printing Machines

- Even if a machine is not given a handle to its own code on its tape at the outset, there are ways for it to construct it.

- Such programs are now called "Quines"



Willard Van Orman Quine 1908-2000

A Java Quine
http://www.knet.ro/lsantha/

```
class Q{public static void main(String[]v){char
c=34;System.out.print(s+c+s+c+';'+'}');}static String
s="class Q{public static void main(String[]v){char
c=34;System.out.print(s+c+s+c+';'+'}');}static String
s=";}
```

- javac Q.java
- java Q

class Q{public static void main(String[]v){char
c=34;System.out.print(s+c+s+c+';'+'}');}static String
s="class Q{public static void main(String[]v){char
c=34;System.out.print(s+c+s+c+';'+'}');}static String
s=";}

C and C++ Quines (authors unknown)

```
char f[] =
"char f[] =%c%c%s%c;%cmain() {printf(f,10,34,f,34,10,10);}%c";
main() {printf(f,10,34,f,34,10,10);}
```

```
#include <iostream>
#define a(b) std::cout<<"#include <iostream>\n#define a(b) "<<#b<<"\nmain(){a("<<#b<<");}"
main(){a(std::cout<<"#include <iostream>\n#define a(b) "<<#b<<"\nmain(){a("<<#b<<");}");}
```

rex Quine
(by a Pomona College Student)

```
a="\\";aa="a";
b="\"";bb="b";
c="=";cc="c";
d="print(

            aa,c,     b,   a,a,b,     f,
           aa,aa,    c,b,aa,b,f,g,bb,c,b,
          a,b,b,f    ,bb,bb,c,b,bb,b,f,g,
         cc,c,b,c   ,b,f,  cc,cc,c,b,cc,
        b,   f,g  ,dd              ,c,b,
       d,    b,f  ,               g,g,
      dd,    dd,                  c,b,
     dd,     b,f                ,g,ee
    ,c,b      ,e,              b,f,
    ee,       ee,              c,b,
   ee,b,f,    g,ff,c,          b,f,
   b,f,ff,ff,c,b,ff,b,f,      g,gg,
            c,b,a         ,nn,b,
             f,gg        ,gg,c,b,
            gg,b,f,      g,nn,nn,c
           ,b,nn,b,     f,g,g,d,g);";
```

continued next col.

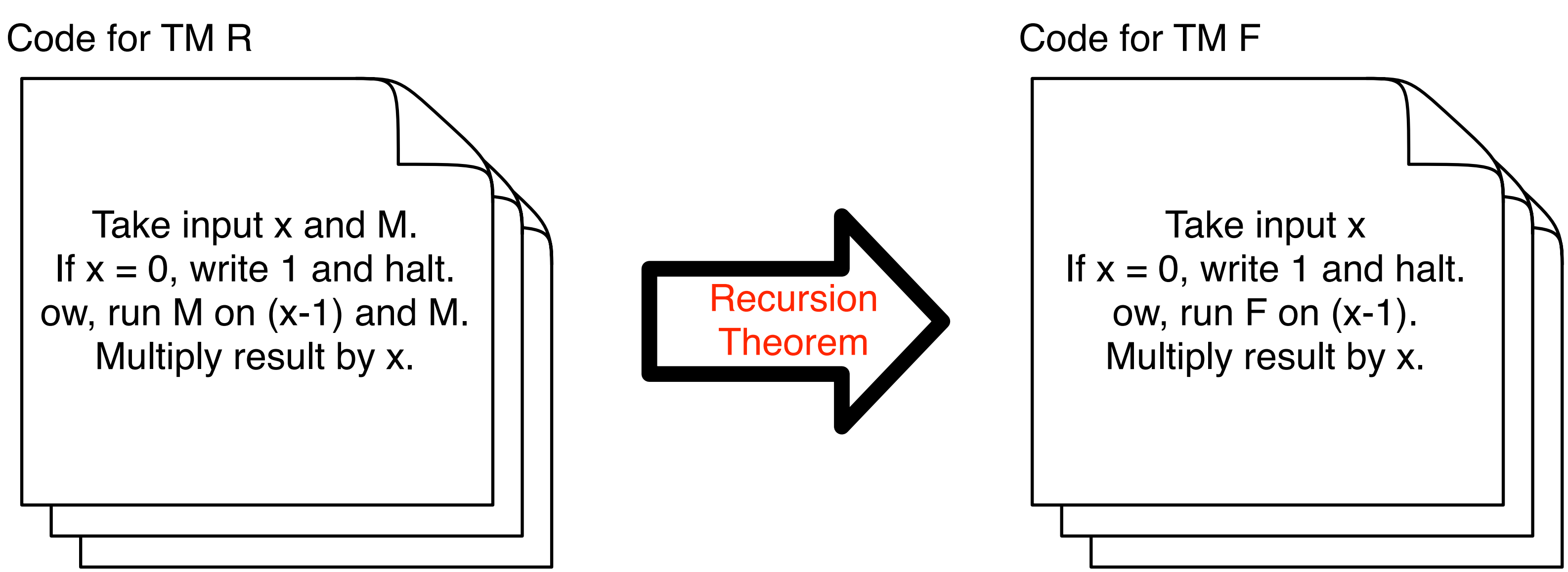
```
dd="d";
e=")";ee="e";
f=";";ff="f";
g="\n";gg="g";
nn="n";

print(

            aa,c,     b,   a,a,b,     f,
           aa,aa,    c,b,aa,b,f,g,bb,c,b,
          a,b,b,f    ,bb,bb,c,b,bb,b,f,g,
         cc,c,b,c   ,b,f,  cc,cc,c,b,cc,
        b,   f,g  ,dd              ,c,b,
       d,    b,f  ,               g,g,
      dd,    dd,                  c,b,
     dd,     b,f                ,g,ee
    ,c,b      ,e,              b,f,
    ee,       ee,              c,b,
   ee,b,f,    g,ff,c,          b,f,
   b,f,ff,ff,c,b,ff,b,f,      g,gg,
            c,b,a         ,nn,b,
             f,gg        ,gg,c,b,
            gg,b,f,      g,nn,nn,c
           ,b,nn,b,     f,g,g,d,g);
```

Applications of Quines

- Entertainment
- Computer viruses
- Artificial life?

Recursion from the Recursion Theorem

- If you have $\langle M \rangle$ on the tape, you can run it.
- A machine M can put $\langle M \rangle$ on the tape.
- Therefore, a TM can compute "recursively" in the modern sense.



Theorem (Sipser)

- The language MIN = $\{ \langle M \rangle \mid M \text{ is minimal} \}$ is not even recognizable.

Minimal

=

no machine with the same behavior has smaller description.

Proof

- Suppose MIN **were** recognizable, hence enumerated by some machine E.
- Consider the TM C:
  - Take input x.
  - Start running enumerator E.
  - Stop when we produce a $\langle D \rangle$ that is **strictly longer** than $\langle C \rangle$ (This must happen. Why?)
  - Simulate D running on x.
- C cannot exist (why?) , so E cannot exist.
- Thus MIN cannot be recognizable.

Other Models of Computation

Two-Stack Machine

- PDA with 2 stacks
  - Transition can push or pop one stack
  - Can transition based on the top symbol of one stack.

Theorem:
A Two-Stack Machine can simulate a TM

- Proof idea:
  - one stack to hold the symbols to the left of the head
  - one stack to hold the symbols below and to the right.
  - Moving the head corresponds to copying a symbol from one stack to the other.

Counter Machines (CM)

- A counter machine is like a TM or PDA, but instead of having a stack or tape, it has one or more integer counters (unbounded capacity)
  - Transitions can increment or decrement one counter
  - Transitions can test whether a counter is zero/nonzero
- Equivalent: PDA with one or more stacks, each having only one symbol (except $ at the bottom)
  - All that matters is the depth of each stack
  - Can only see the top of stack, so can only test empty/nonempty
- NB: Any language accepted by a CM is recognizable.

Theorem:
A 3-counter machine can simulate a TM.

- **Idea**: A 3-counter machine can simulate a 2-stack machine.
- Suppose our 2-stack machine uses r different stack symbols. Wlog, call these symbols 0 .. r-1.
- Represent the stack $X_1X_2 \cdots X_n$ (with X_1 on top) by the number

$$X_nr^{n-1} + X_{n-1}r^{n-2} + \cdots + X_2r + X_1$$

  - Pop = divide by r; top symbol is the remainder.
  - Push k = multiply by r and add k.
- Two of the counters hold two (encoded) stacks.
- Arithmetic operations use the third counter as scratch space

Theorem:
A 2-counter machine can simulate a TM.

- **Idea**: A 2-counter machine can simulate a 3-counter machine.
- Represent the 3 counters i, j, and k by the single number

$$2^i 3^j 5^k$$

- All necessary arithmetic can be done using the second counter as scratch space.

Partial Recursive Functions

- Purely mathematical formulations of functions of natural numbers.
  - Small set: Primitive Recursive functions (total)
  - Larger set: Partial Recursive functions (possibly partial)
- Theorem:
  - A TM can compute exactly the partial recursive functions (partial because TM might not terminate on all inputs)
  - Partial Recursive functions can (by encoding tapes as integers) simulate any TM.

Primitive Recursive Functions (an inductive definition!)

- The constantly-zero functions
  (take k ≥ 0 arguments; return 0)

- The projection functions
  (take k ≥ 0 arguments; return the i^{th})

- The successor function
  (take one argument n; return n+1)

Primitive Recursive Functions (continued)

- The composition of prim. rec. functions is prim rec.
  e.g., h(x,y) = f(g1(x,y), g2(x,y), g3(x,y))

Corollary: the constantly-7 functions is PRF, for any n, because they are compositions of the constantly-zero function and sucessors.

Note: we often write explicit definitions like

h(x,y) = f(g1(y), g2(x), 2)

rather than

h(x,y) = f(g1(proj2(x,y)), g2(proj1(x,y)), succ(succ(zero())))

Primitive Recursive Functions (continued)

- If b and r are prim rec. (with appropriate # of arguments) then so is the function f defined by

$$f(0, x_1, \ldots, x_n) \quad = b(x_1, \ldots, x_n)$$

$$f(n+1, x_1, \ldots, x_n) = r(x_1, \ldots, x_n, n, f(n, x_1, \ldots, x_n))$$

Note: this "primitive recursion" is a very stylized (restricted) template for coding recursively.

Examples of Primitive Recursive Functions

- plus(x, y)
- times(x, y)
- pred(x)
- monus(x, y)
- mod(x, y)
- div(x, y)
- sqrt(x)
- ?:
- Predicates like even, equality, etc. (return 0/1)

```
plus(0, y) = y
plus(n+1, y) = succ(plus(n,y))
```

```
mult(0, y) = 0
mult(n+1, y) = plus(y, times(n, y))
```

```
fact(0) = 1
fact(n) = mult(n, fact(n-1))
```

Primitive Recursion

- **Theorem**: Primitive Recursive functions are total.
- **Proof**: By induction.

- Are all total functions TM-computable?
- Are all TM-computable total functions primitive recursive?

Computable but not Primitive Recursive: Diagonalization

- We (hence a TM) can enumerate all the primitive recursive functions $p_1, p_2, p_3, \ldots$
- Then $q(n) = p_n(n) + 1$ is total and computable.

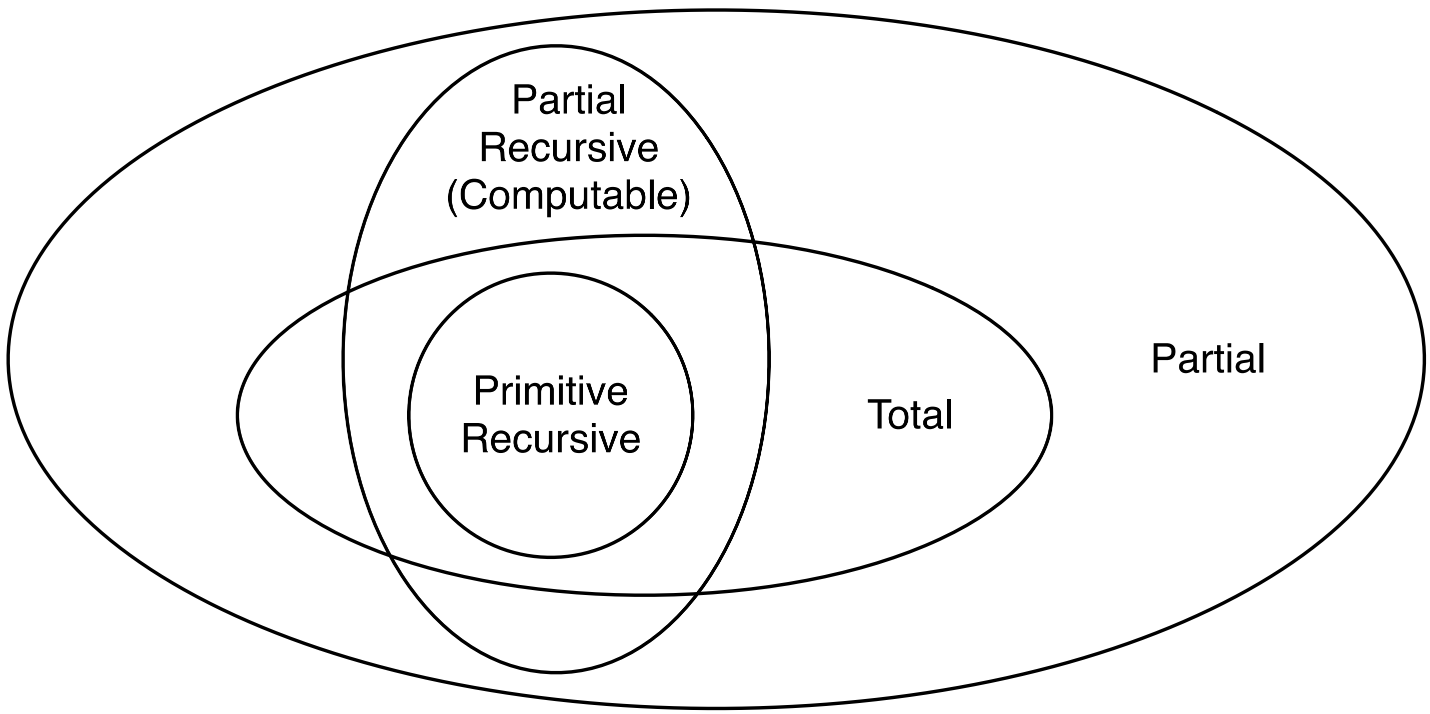
Computable but not Primitive Recursive: Ackermann Hierarchy

$$\boxed{\begin{aligned} A_0(m) &= S(m) \\[1em] A_{n+1}(0) &= A_n(1) \\ A_{n+1}(m+1) &= A_n(A_{n+1}(m)) \\[1em] A(m) &= A_m(m) \end{aligned}}$$

Theorem:
A(m) is a total function that grows faster than any primitive recursive function of one argument.

$$A(1) = 3$$
$$A(2) = 7$$
$$A(3) = 61$$
$$A(4) = 2^{2^{2^{65536}}} - 3$$

Primitive Recursive vs. Partial Recursive Functions



(not to scale)

Partial Recursive Functions

- Prim. Rec. + a partial "minimization operator"

If $h(x_1, \ldots, x_n)$ is partial recursive, then

$$\mu k\ [h(k, x2, \ldots, x_n) = 0]$$

is the function that (given $x_2, \ldots, x_n$) computes the least k (if any) that makes $h(k, x2, \ldots, x_n)$ zero.

Examples

$$\mathrm{sqrt}(n) = \mu k.\ [\mathrm{monus}(n, \mathrm{times}(k, k)) = 0]$$

$$\mathrm{diverge}(n) = \mu k.\ [\mathrm{monus}(k+1, k) = 0]$$

$$\mathrm{strange}(m, n) = \mu k.\ [\mathrm{not}(\mathrm{eq}(k+m, n)) = 0]$$

Partial Recursion vs. TMs

- Every Partial Recursive operation (including minimization) could be programmed in a TM.

TMs vs. Partial Recursion

- We can encode TM configurations as numbers.
- Primitive Recursive functions exist
  - $R(x)$ = configuration one step beyond configuration x.
  - $T(i,x)$ = configuration i steps beyond configuration x.
  - $P(x)$ = whether configuration x is halting (0 or 1)
- Computation length = $\mu i.\ [P(T(i, x_0)) = 0]$
- Final configuration = $T(\mu i.\ [P(T(i, x_0)) = 0], x_0)$

Enumerating Partial Recursive Functions

- For each k, the Partial Recursive Functions with k arguments can be enumerated $f_1, f_2, \ldots$
  - There is even a k+1-argument PRF that given i, computes f_i applied to the k remaining arguments.
- Why doesn't the diagonal argument work again, giving us a computable function not in this list?
- $\{ j \mid f_j(\bullet) \text{ is total } \}$ is not recognizable/enumerable.
- $\{ j \mid f_j(j) \text{ is defined } \}$ is recognizable, not decidable.